



Europäisches
Patentamt

European
Patent Office

Office européen
des brevets

PT1F99635
LIS

Bescheinigung

Certificate

Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. Patent application No. Demande de brevet n°

99403326.4

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

I.L.C. HATTEN-HECKMAN

DEN HAAG, DEN
THE HAGUE, 23/10/00
LA HAYE, LE

THIS PAGE BLANK (USPTO)



Europäisches
Patentamt

European
Patent Office

Office européen
des brevets

Blatt 2 der Bescheinigung
Sheet 2 of the certificate
Page 2 de l'attestation

Anmeldung Nr.:
Application no.: **99403326.4**
Demande n°:

Anmeldetag:
Date of filing: **30/12/99**
Date de dépôt:

Anmelder:
Applicant(s):
Demandeur(s):
Koninklijke Philips Electronics N. V.
5621 BA Eindhoven
NETHERLANDS

Bezeichnung der Erfindung:
Title of the invention:
Titre de l'invention:
Multi-tasking software architecture

In Anspruch genommene Priorität(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)

Staat: Tag: Aktenzeichen:
State: Date: File no.
Pays: Numéro de dépôt:

Internationale Patentklassifikation:
International Patent classification:
Classification internationale des brevets:

Am Anmeldetag benannte Vertragstaaten:
Contracting states designated at date of filing: AT/BE/CH/CY/DE/DK/ES/FI/FR/GB/GR/IE/IT/LI/LU/MC/NL/PT/SE
Etats contractants désignés lors du dépôt:

Bemerkungen:
Remarks:
Remarques:

THIS PAGE BLANK (USPTO)

MULTI-TASKING SOFTWARE ARCHITECTURE.**FIELD OF THE INVENTION**

The invention relates to computer systems and software architectures for computer systems. More particularly, it deals with a software architecture comprising semaphore means for stalling a task until one of a predetermined set of events occurs.

5

BACKGROUND ART

The European patent application published under number 592 117 A2 describes an asynchronous inter-process communications arrangement in an operating system for real time and multiprocessor applications.

10

SUMMARY OF THE INVENTION

It is an object of the invention to provide a computer program product for real time applications using a cheap operating system suitable for use, for example, in embedded systems.

15

In accordance with the invention, a computer program as defined in the opening paragraph is characterized in that the computer program comprises specific message files associated with said first task for receiving data to be processed by the first task, the occurrence of one of said set of predetermined events causing a piece of data to be written in one of the associated messages files. The consequences of the predetermined events may include for example, a writing in one of the specific message files from another task or from an interruption handler. The later can be triggered by a hardware interruption (IT) which causes the current task to be stopped or "interrupted" by an interruption handler for the processing of the hardware interruption. The computer program in accordance with the invention can be implemented using any operating system having basic synchronization tools. Such synchronization tools include semaphore means for causing a task to be waiting on a unique event such as the release of the semaphore by another task. They also include blocking mechanisms for temporarily blocking "interruptions" coming from interruption handlers during a data writing (message reception) in the message files associated with the first task. The word "interruption" is put into brackets so as to distinguish an "interruption" caused by an interruption handler from the hardware interruption which is to be handled by the interruption handler and which is the source of the software "interruption". Blocking

mechanism for temporarily blocking "interruptions" from interruption handlers allow to prevent a memory conflict during the storage of the data corresponding to the message to be sent to the first task. Embedded systems are preferably implemented using simple operating systems (OS). Such OS generally do not comprise sophisticated synchronization tools such as "select" means for causing the execution of a task to be dependent on several events coming from different entities such as tasks and IT handlers. In contrast, complex OS generally comprise a "select" mechanism using file descriptors allowing to stall a task (that is to say to send a task to sleep) until a predetermined event occurs. A data writing in one of the files is an example of event. This implies that the OS comprises file handling means which are rather complex. Thus they are not appropriate in embedded systems. The invention, in effect, provides an equivalent to the "select mechanism" but which uses only very basic synchronization tools so as to be suitable for embedded systems.

An other advantage of the invention relates to the following aspects. An interrupt handler, which has very poor capabilities compared to a task, can directly write into a message file of the type, for example, of a simple FIFO (first in first out) data structure. Since writing data in such a message file substantially requires pointer manipulations, this can be performed by an interruption handler (as well as by a task). In complex OS using file descriptors it is not possible for an IT handler to write into a file. In this case actually, a data writing requires actions which are much more complex than pointer manipulations so that they can not be performed by an interruption handler. By using a simple data structure, called message file, associated to a task, the invention allows a task to receive messages directly from interruption handlers as well as from other tasks, which is very interesting.

The invention and additional features, which may be optionally used to implement the invention to advantage, are apparent from and will be elucidated with reference to the drawings described hereinafter.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a conceptual diagram for illustrating the invention.
Fig. 2 is a block diagram of a computer system according to the invention.
Fig. 3 is a block diagram of an electronic apparatus according to the invention.
Fig. 4 illustrates a method according to the invention.

DETAILED DESCRIPTION OF THE DRAWINGS

Fig. 1 illustrates a computer program product according to the invention having a software architecture for synchronizing a first task F1 with respect to a set of predetermined events. The predetermined events may cause the execution of tasks 5 denoted F2 and F3 or of interruption handlers (or IT handlers) denoted IT1 and IT2. The tasks and IT handlers are carried out by a microprocessor as illustrated in Fig. 2. The IT handlers handle hardware interruptions and the tasks are executed by the microprocessor according to a program.

Message files denoted MF1 to MF3 are associated with the first task F1 for 10 receiving messages caused by the occurrence of the predetermined events. As shown in Fig. 1, the IT handlers IT1 and IT2 are reserved a message file MF1 and the tasks F2 and F3 are reserved the message files MF2 and MF3 respectively. According to a preferred embodiment of the invention, priority levels can be assigned to the message files. The message files with higher priority level would be carried out prior to the 15 message files with lower priority levels. Preferably, the files associated to interrupt handlers that is to say the file MF1, are given a higher priority level than the files associated to tasks that is to say MF2 and MF3.

A message file can be a simple data structure, such as a FIFO for example, 20 wherein data can be stored and retrieved only by manipulating pointers, that is to say very quickly. The FIFO is a non-limitative example. A great many other data structures can be used for implementing the message files, provided they can be accessed by a IT handler as will be described herein after. A writing in one of these message files can be done from an IT handler IT1 or IT2 as well as from a task F2 or F3. Because such writing requires only to manipulate pointers on memory tables for memory allocation, it 25 only takes a few microseconds. It is thus possible for an IT handler, which has very poor functions, to cause a data writing in these message files. Without such a data structure associated to the first task, it would be necessary to have a specific extra task for processing messages from IT handlers, the messages being destined to the first task.

The architecture comprises semaphore means denoted S for stalling the first 30 task F1 (it is said that the task is pending on the semaphore S) until one of the predetermined set of events occurs. The event can cause a writing in one of the message files from a IT handler or a task. The writing also causes a release of the semaphore. The semaphore means comprise a counter that is incremented each time 35 one of said predetermined set of events occurs. The first task F1 is thus pending on its semaphore until one of the predetermined set of events occurs. The occurrence of the

event wakes the task up for executing a specific program with respect to the occurred event. The semaphore's counter is incremented upon reception of a message in any of the associated files. The counter indicates how many messages are not read yet by the first task F1. Each time a message is read by the first task F1, the counter is
5 decremented.

All received messages are actually physically stored in a pre-allocated memory table or file. A memory location chain is allocated to each message file and contains all the received messages of each file. The information contained in the messages can be retrieved by accessing the address of the first location of a chain.

10 Upon reception of a new message, a free location is allocated for the new message. In order to avoid that a free location is allocated for two different purposes at the same time, it is necessary to temporarily block "interruptions" during reception of the message to be sure that allocation can be complete. A blocking mechanism is generally comprised in the operating system. It is used here for temporarily blocking
15 "interruptions" coming from the IT handlers IT1 and IT2 during a message reception, that is to say during a data writing in one of the message files. In effect, blocking the interruptions prevents the tasks or IT handlers from taking priority over processing of the current task. As mentioned before, the memory allocation takes very little time. The
20 interrupts then need to be blocked for a few microseconds only which is generally acceptable for real time applications.

While no new message is received, the first task is pending on its semaphore S until an event occurs from one of the tasks or IT handlers. This is illustrated by an arrow 11 in broken lines starting with a point. An occurrence of an event causes a certain data to be written into the reserved file, which is illustrated with a plain arrow
25 12. The task or IT handler which caused said data writing, also caused the semaphore S to be released and its counter to be incremented. This indicates that a new message is received in one of the associated message files. The release of the semaphore is indicated by an interrupted arrow 13. Upon release of the semaphore S, the first task F1 is waken up so as to resume its execution.

30 According to the preferred embodiment of the invention, as soon as the first task is waked up, it starts looking into the message files with the highest priority level to check if there are unread messages in this message file. It then reads and processes the messages in the order of priority given to the files in which the messages are received. As shown in Fig. 1, the task F1 starts reading the messages in the message
35 file MF1 which has the highest priority. Once a message is read, the semaphore's

device, or set top box, destined to receive cable or satellite audio and video interactive programs.

The electronic apparatus illustrated in Fig. 3 comprises a parabolic antenna 31 including a frequency conversion block, a tuner TUN and an intermediate frequency 5 amplifier IF. The intermediate frequency amplifier IF provides an output signal which is converted into digital values in an analogue to digital converter A/D and is subsequently demodulated by a demodulator DEM. Finally, certain errors are corrected in an error corrector F.E.C. ("Forward Error Correction").

The description given below is a non-limitative example: the input of the device 10 might just as well be connected to, for example, a cable or a hertzian distribution system, in which cases the modules TUN, IF, A/D and DEM would be different.

The demodulated signal at the output of the error corrector F.E.C. is an MPEG-2 TS signal ("MPEG-2 Transport Stream"). This signal comprises several programs with sound and data, which may be scrambled in some cases.

15 In the simplest case, this MPEG-2 TS signal is directly supplied through a connection 32 of the parallel type to an unscrambling, conditional access, and demultiplexing integrated circuit 33. The signal is unscrambled there in an unscrambling circuit DSC, transmitted to a microprocessor μ P via a bus, and then to a demultiplexer DEMUX via a bus too. The demultiplexer selects a program from the 20 MPEG-2 TS signal. This program is applied in digital form to a video decoder 34 and to an audio decoder 35. In response, these decoders supply analog signals for projecting an image on a screen and generate sound from a loudspeaker, respectively. These decoders are followed by audio and video circuits 36, 37 respectively, connected to an external television. The device described here may thus be a box which is designed to 25 be connected to an input of a peripheral TV monitor or a classical TV set. Alternatively, all these circuits may be incorporated in a TV set.

A software architecture as shown in Fig. 1 can be implemented, for example, in 30 a microprocessor bloc of the unscrambling, conditional access, and demultiplexing integrated circuit 33. An interrupt IT1 or IT2 can be caused by, for example, the presence of an ASCII character on the serial line (not shown on the figure) for testing purposes during the manufacturing process. An interrupt may also be caused by the presence of a RC6 remote control code during the normal working process of the set top box. An interrupt may also be caused by the presence of a data burst at the entry 35 of the demultiplexer DEMUX. By way of example, the first task F1 can be waiting for a TCP/IP stack coming from the other tasks F3 or F2.

counter is decremented. The reading is illustrated by an arrow in solid lines starting with a point 14.

If the semaphore's counter is positive (or equal to zero depending on the implementation of the semaphore in the operating system), there are unread messages 5 in at least one of the associated message files. In that case, the first task keeps on reading and processing the received messages. As soon as the counter becomes negative (or equal to zero in the case that zero is not included in the preceding), it is said that the first task is blocked on its semaphore until a new message arrives and causes the counter to become positive again. The same mechanism as described with 10 the first task F1 can be implemented with the tasks F2 and F3.

The mechanism described herein before causes a task, the first task F1 as shown in Fig. 1, to be stalled until one of a predetermined set of events occurs. These events may be caused by execution of tasks, like F2 and F3, or by execution of 15 interrupt handlers, like IT1 and IT2. The mechanism is based on the implementation of the message files which are associated to the first task for receiving messages from other tasks or from IT handlers. The interrupts are temporarily blocked during the writing of the messages to prevent a memory conflict.

Fig. 2 is a block diagram illustrating a computer system for implementing a software architecture as shown in Fig. 1. It comprises a microprocessor uP which is 20 embedded in an apparatus. The microprocessor executes a pre-stored program so as to control the apparatus. The program includes tasks 21 which are controlled by a scheduler SCHED with respect to the program. Interrupts denoted ITs may be generated by hardware components. The arrival of an ASCII character on the serial line may cause an interrupt. The interrupts are first processed by an interrupt server IT 25 SERV. It selects an appropriate interrupt handler 25 depending on the source of the interrupt. Subsequently, the IT handler processes the interrupt. In fig. 2, the IT handlers and tasks are represented inside the microprocessor bloc. This only means that they are executed by the microprocessor. However, the code associated to the tasks and interrupts can be stored in any memory location either inside or outside the 30 microprocessor. Specific memory means are also indicated in the form of a bloc MEM for representing the memory locations corresponding to the message files MF1, MF2 and MF3 or the semaphore means as described with reference to Fig. 1. The message files and semaphore means can also be stored in separate memory means.

A block diagram of an example of an electronic apparatus according to the 35 invention is illustrated in Fig. 3. It diagrammatically shows a TV receiving and decoding

An advantageous application of the invention is described below. According to this example, the predetermined events causes writings in specific message files associated to a first task, by other tasks or interrupts handler. In the latter case, the writing is caused by a hardware interruption which is to be handled by an appropriate 5 IT handler. The IT handler may need to send a message to a specific task. For example, if the interruption is related to the arrival of an ASCII character, the IT handler which handles this interruption may need to communicate the ASCII character to the first task. Normally, it would be necessary to have a specific task for transferring the message from the IT handler to the destined first task. According to the invention 10 which provides associated message files to the tasks, the IT handler can send its message, containing the ASCII character, to the destined first task via a reserved message file which is associated to the first task. The first task can collect all ASCII characters coming from the IT handler and build a character chain. Subsequently, the first task can transmit the whole chain to a second task which needs to process this 15 character chains.

Fig. 4 is a diagram for illustrating a method of synchronizing a task with respect to an occurrence of one of a predetermined set of events, denoted E, according to the invention. The steps of the method are indicated in cases K0 to K4, wherein :

- K0 is a waiting step WAIT for making a first task F1 wait on a specific semaphore until 20 one of a predetermined set of events E occurs, the first task being associated with message files for receiving messages caused by the occurrence of the predetermined events,
- K1 is a blocking step "block" for temporarily blocking interruptions coming from IT 25 handlers during reception of said messages in order to avoid memory allocation conflicts during the following message reception step,
- K2 is a message reception step "recept" for receiving a message in one of the associated files of the first task,
- K3 is a wake up step "wake" for waking up the first task upon the occurrence of one 30 of the predetermined event,
- K4 is a reading and processing step "R&P" for reading and processing the messages received in the message files.

THIS PAGE BLANK (USPTO)

Claims.

1. Computer program product comprising semaphore means for stalling a first task until one of a predetermined set of events occurs,
characterized in that the computer program product comprises specific message files associated with said first task for receiving data to be processed by the first task, the occurrence of one of said set of predetermined events causing a piece of data to be written in one of the associated messages files.
5
2. Computer program product as claimed in claim 1, wherein said set of predetermined events includes hardware interruptions, a hardware interruption causing a selected interruption handler to write data into one of the message files associated with the first task, a blocking mechanism being used for temporarily blocking the interruptions during said data writing into the message file.
10
3. Computer program product as claimed in any of claim 1 or 2, wherein said set of predetermined events includes at least an event causing a second task to write data into one of the associated message files.
15
4. Computer program product as claimed in any of claim 1 to 3, wherein said message files are given a priority level, for the first task to process the data received in the associated files in the order of priority of the file in which the data are received.
20
5. Computer program product as claimed in claim 4, wherein the message files with the highest priority level are allocated to interruption handlers so as to receive data from said interruption handlers.
25
6. Computer program product as claimed in any of claim 1 to 5, wherein said message files comprise a data structure which is accessible by pointer manipulation.
7. Computer system for implementing a computer program as claimed in any of claim 1 to 6, wherein said system comprises :
 - a processor for executing a set of schedulable tasks including at least the first task,
 - a scheduler for selecting one of said schedulable set of tasks for execution by said processor,
 - at least an interrupt handler for handling interrupts,

- an interrupt server for performing a specific function in response to the occurrence of an interrupt.

8. Electronic apparatus comprising a computer system as claimed in claim 7.

9. Method of synchronizing a first task with respect to an occurrence of one of a 5 predetermined set of events, comprising :

- a waiting step for making said first task wait on a specific semaphore until one of said predetermined set of events occurs, said first task being associated with message files for receiving data in a message, said data reception being caused by the occurrence of one of said predetermined events

10 - a blocking step for temporarily blocking interruptions during reception of said data,

- a reception step for receiving said data in one of the associated message files,

- a wake up step for waking up said first task upon reception of the data in one of its associated files,

- a reading and processing step performed by said first task for reading and processing 15 the data received in a message file.

10. Method as claimed in claim 9, wherein said set of predetermined events includes hardware interrupts causing interrupt handlers to write data into one of the message files associated with the first task.

Abstract.

The invention relates to a computer program product comprising semaphore means for stalling a first task until one of a predetermined set of events occurs. The computer program product comprises specific message files associated with said first task for receiving data to be processed by the first task. The occurrence of one of said 5 set of predetermined events causing a piece of data to be written in one of the associated messages files. The computer program in accordance with the invention can be implemented using any operating system having basic synchronization tools. Such synchronization tools include semaphore means for causing a task to be waiting on a unique event such as the release of the semaphore by another task. They also include 10 blocking mechanisms for temporarily blocking "interruptions" coming from interruption handlers during a data writing (message reception) in the message files associated with the first task. The invention, in effect, provides an equivalent to the "select mechanism" but which uses only very basic synchronization tools so as to be suitable for embedded systems.

15

Fig. 1

THIS PAGE BLANK (USPTO)

1/3

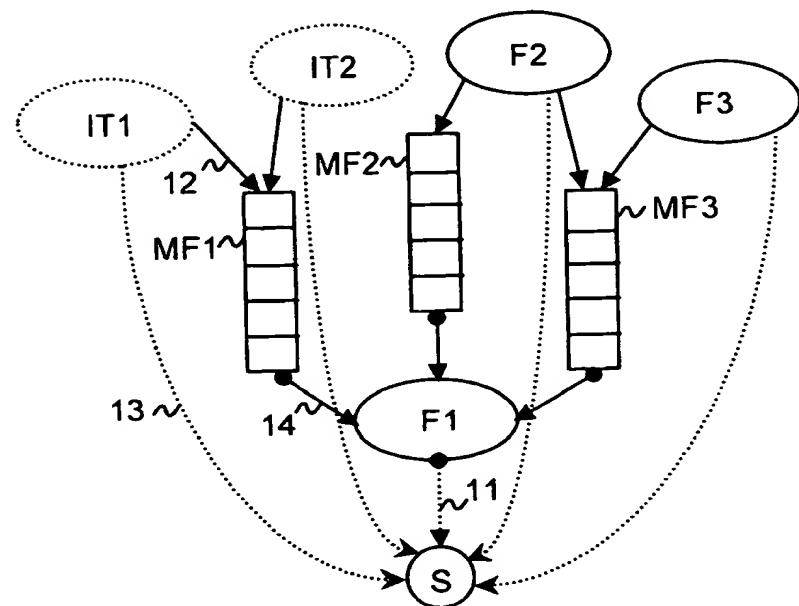


FIG.1

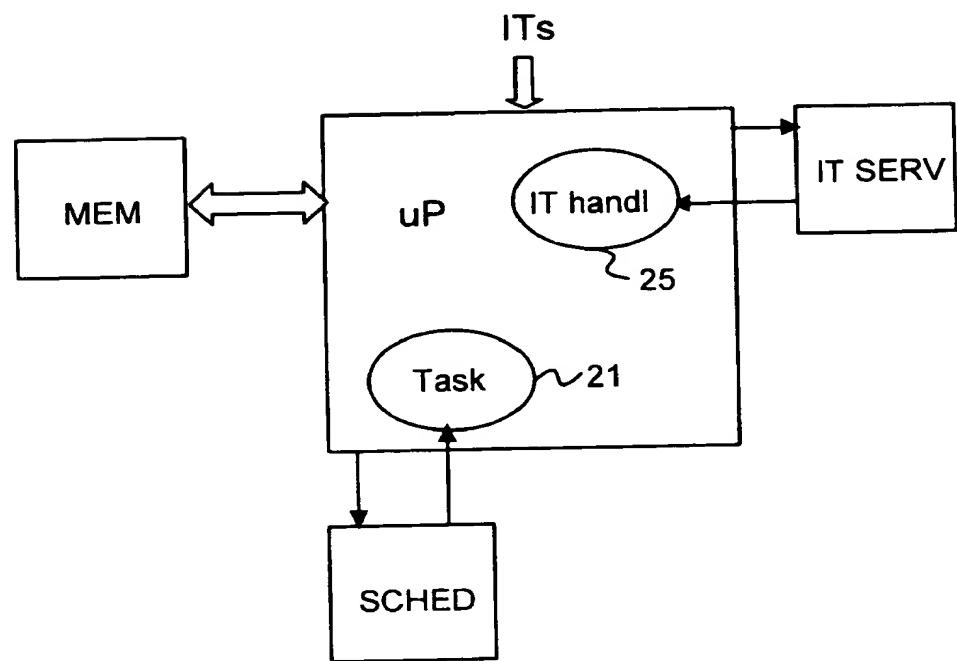


FIG.2

1-III-PHF99635

2/3

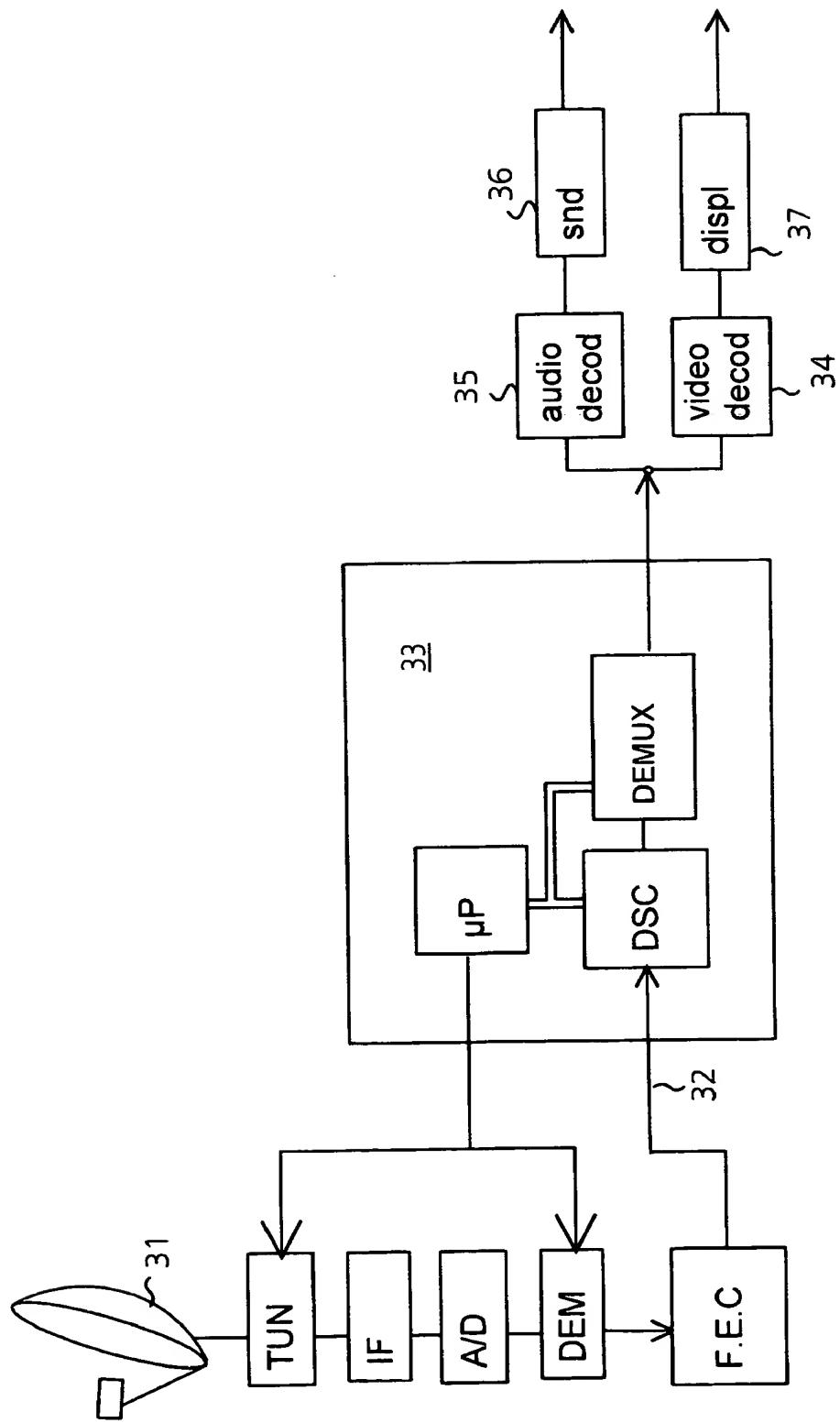


FIG. 3

2-III-PHF99635

3/3

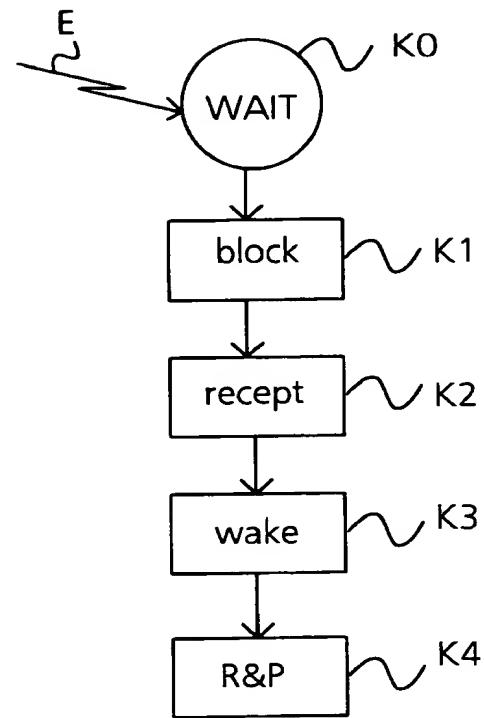


FIG.4

3-III-PHF99635

THIS PAGE BLANK (USPTO)